This document contains the accepted but unedited version of the following publication:
Feedback control by online learning an inverse model. Tim Waegeman, Francis wyffels and Benjamin Schrauwen. IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, Vol. 23(10), pp. 1637-1648 (2012)

The abstract can be found on the IEEE Xplore web site:
http://dx.doi.org/10.1109/TNNLS.2012.2208655

Feedback Control by Online Learning an Inverse Model

Tim Waegeman, Francis Wyffels, Benjamin Schrauwen*

***Abstract*—Often a model, predictor or error estimator is used by a feedback controller to control a plant. Creating such a model is difficult when the plant exhibits nonlinear behavior. In this paper a novel online learning control framework is proposed which does not require explicit knowledge about the plant. This framework uses two learning modules, one for creating an inverse model, and the other for actually controlling the plant. Apart from their input, they are identical. The inverse model is learning by the exploration performed by the not yet fully trained controller, while the actual controller is based on the currently learned model. The proposed framework allows fast online learning of an accurate controller. The controller can be applied on a broad range of tasks with different dynamical characteristics. We validate this claim by applying our control framework on several control tasks: the heating tank problem (slow nonlinear dynamics), flight pitch control (slow linear dynamics) and the balancing problem of a double inverted pendulum (fast linear and nonlinear dynamics). The results of these experiments show that fast learning and accurate control are achieved. Furthermore, a comparison is made with some classical control approaches and observations concerning convergence and stability are made.**

***Index Terms*—Adaptive control, feedback control, neural network (NN), Reservoir Computing (RC), heating tank, pitch control, inverted pendulum.**

I. Introduction

DYNAMICAL systems (DS) are everywhere: in organisms, cyclic natural phenomena but also in man-made systems like thermostats, planes, robotics, ... Man-made systems are often referred to as plants and are modeled as having an output and an input. As shown in Fig. 1, feedback controllers, such as the one proposed in this work, use the feedback of the dynamical system (plant-output), compared with a desired plant-output, to control the plant-input. For instance, the cruise control of a car uses a feedback controller to keep the velocity constant. However, when the car is driving downhill the car will go faster because of gravitation. The controller observes this increase in velocity and reduces the throttle to ultimately converge to the desired velocity.

Several standard control algorithms use pre-acquired knowledge about a system to accomplish the desired behavior. The control of linear systems has been extensively studied [1], [2]. However, more complex and nonlinear systems are hard to model correctly. One approach to solve this problem is to use a learning approach such as using a Neural Network (NN) [3]. In [4], a NN was used to change the motor commands by predicting the possible errors of a movement. Since the publication of [5], the use of NNs for identification and control of nonlinear systems has gained a lot of interest. For instance, Nguyen and Widrow [6] used a NN to control the truck backer-upper problem. In [7] and [8] NNs are used in combination with a classical Sliding Mode Control approach. Other approaches such as [9] and [10] use a NN to train a predictor which is used to construct output feedback control. In these works, NNs are used as static function approximators or, when their input is a tapped delay line, to model a finite memory functional dependence.

A more natural and richer alternative to using tapped delay lines is by allowing recurrent connections in the the NN, then called Recurrent NNs (RNNs). These have very successfully been used to control nonlinear dynamic systems [11], [12], [13]. In recent work [14], Prokhorov superbly demonstrates the very rich modeling capabilities of RNNs in a neurocontroller for the electric throttle of a hybrid vehicle. Although several neurocontrollers incorporate some prior knowledge, some approaches achieve control without any prior information about the plant. For instance in [15] and [16], an adaptive neural controller is used to embed the unknown system dynamics of a control process. In [17], two RNNs are used: one for approximately modeling the nonlinear plant, the other to control towards the desired system response. However, RNNs are notoriously difficult to train due to the problem of fading gradients calculated with Backpropagation Through Time and the regular bifurcation encountered during training using stochastic gradient descent [18], [19], [20].

A solution to the problem of training RNNs is proposed by the Reservoir Computing (RC) paradigm, which unifies a set of similar techniques to efficiently train RNNs. The core idea is that only the output weights of the network are trained, and that the internal, recurrent connections are randomly initialized such that the dynamics of the network are at the edge of stability [21].

To avoid the need of prior knowledge, we propose an online feedback control framework which learns to control a plant by online leaning an inverse plant model based on real-time controlled plant-input output pairs. In parallel, this preliminary model is used to actually control the system. Because the initial controller is not optimal, small mistakes are made which can be seen as exploration that allows the system to learn a better plant model, leading to a better controller. Experiments demonstrate the ability to learn the control of plants, with a wide variate of dynamics, both online and quickly. As a result, our control framework can be applied on a wide variety of applications.

In this work, we will use RC networks as basic learning modules, but the framework itself is general. Any machine learning technique which is able to model temporal function-

*Department of Electronics and Information Systems, Ghent University, Ghent Belgium.
Tim.Waegeman@UGent.be, Francis.Wyffels@UGent.be and Benjamin.Schrauwen@UGent.be

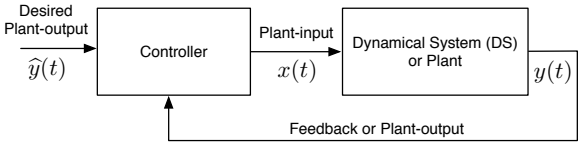

Fig. 1. Illustration of a simple feedback controller and a dynamical system (DS) or plant, accompanied by the used terminology. $y(t)$ and $\widehat{y}(t)$ represent the actual and desired plant output respectively. The output of the controller is denoted by $x(t)$.

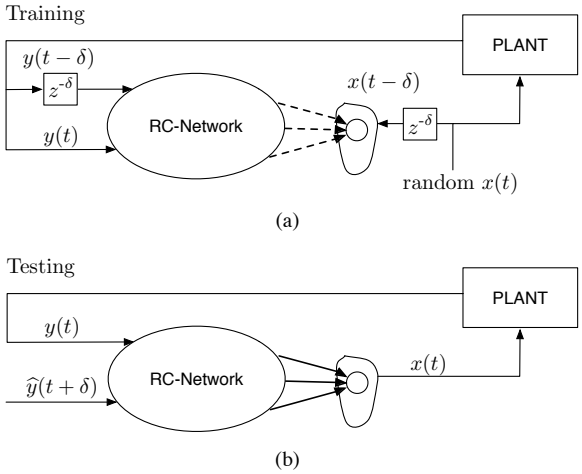

Fig. 2. Illustration of a controller method described in the work of Herbert Jaeger [23]. During training, random $x(t)$ values are used to train the output weights of the network based on the plant response $y(t)$ on these values. Afterwards, during testing, the trained network is used to control the plant according to to the desired plant-ouptput $\hat{y}(t+\delta)$

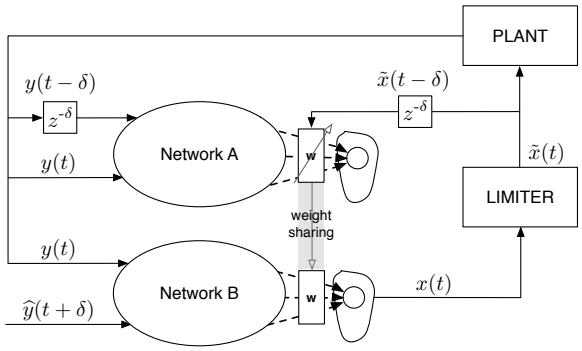

Fig. 3. Schematic representation of the proposed controller. The dashed arrows represent the output weights **w** which are trained. These are the same for both networks (weight sharing). The optional limiter limits the values $x(t)$ to a desired range which, for example, represent imposed motor characteristics. Afterwards, the limited values $\tilde{x}(t)$ excite the plant. The values $\tilde{x}(t-\delta)$ are used as desired network *A* output, which are used to train the weights **w**. The resulting weights are used for network *B* as well.

als online could be used: tapped-delay line models with non-linear regression or neural networks, regular RNNs, Long Short Term Memory RNNs [22], etc.

The remainder of this paper is structured as follows: first, in Section II, we describe the design of our control framework. To demonstrate the abilities of this framework we use a RNN at the core of this framework by applying it according to the Reservoir Computing approach. Therefore, in Section III we give a short introduction on Reservoir Computing and explain the training algorithm in more detail. Next, we analyze the stability and convergence of the obtained controller in Section IV. Afterwards, the controller's performance is evaluated by applying it to different control problems: the heating tank problem, flight pitch control and the rotational double inverted pendulum. In Section V we discuss these experiment in more detail. Finally, we draw our conclusions in Section VI.

II. Design of the Control Framework

Classical approaches of feedback controllers can be grouped by techniques that do or do not use prior knowledge of the plant. The latter, such as PID controllers, use no direct information of the plant dynamics. Other non-prior-knowledge-based techniques use a model exploration strategy where the produced observation of a random action are used by the controller to adjust its control. In the work by Jaeger [23], such a strategy is taken. Here a Reservoir Computing (RC) network is used which is trained offline by using random values as training output and the plant response to these values as training input. In this example, the feedback information $y(t)$ excites the RC-network in 2 versions: the current feedback $y(t)$ and a delayed version $y(t-\delta)$. During training, also the desired output, which are the random plant-input values $x(t)$, are delayed δ time steps before being used as training data of the RC-network.

The main reason for this delayed network input is to allow afterwards a desired plant-output and the current plant feedback as network input. After training the output weights (dashed lines in Fig. 2(a)), the desired plant-output $\widehat{y}(t+\delta)$ is given to the input which was connected to $y(t)$ during training. The actual plant-output on the other hand, is given to the input of the network which was connected with $y(t-\delta)$ during training. As illustrated in Fig. 2(b), the resulting network output $x(t)$ drives the plant-input.

The idea here is to model the progress in plant-input $x(t-\delta)$ given the past and current plant-output ($y(t-\delta)$ and $y(t)$). This model is afterwards used to determine the plant-input given the current and future plant-output ($y(t)$ and $\widehat{y}(t+\delta)$) where the model is expected to generalize the trained behavior.

It should be noted that the choice of a good δ is essential to find such a model. δ and thus the sample rate, determines the amount of time that the used network has to reach the desired plant-output. However, in this work, we assume the used sample rate to be predetermined. Therefore, the effect of δ depends on the dynamics of the plant. A smaller δ is used for a plant with fast dynamics and a larger one for a plant with slower dynamics.

A. Proposed Feedback Control Framework

In order to allow online control in which no prior knowledge (model) is necessary we propose the control framework shown in Fig. 3. Here, a similar network to the one described

above (Fig. 2(a)) is used. This network, called network A, is trained online in a supervised manner by using Recursive Least Squares (RLS). Below network A, we have a duplicate network, network B, with the same input, network and output weights (weight-sharing). This network is connected to the plant in a similar manner as the network in Fig. 2(b). The output of this network is not only connected to the plant but is also used (delayed with δ time steps) as desired output $\widehat{y}(t+\delta)$ for the training of the output weights. The network states are initially the same for both networks and are randomly chosen according to a normal distribution ($\mathcal{N}(0,1)$). This random initialization is necessary to initiate the plant with random values. Without these values the amount of information necessary to train the internal model will be insufficient to generalize well. Because the inputs for both networks are not the same, the corresponding states will evolve differently. However, as network A is converging to a more accurate model, the inputs of both networks will converge to each other with a difference of δ time steps. Because of the desired plant output and the current plant feedback as input, network B starts generating values which are given to the plant. For some plants it might be necessary to limit these values to a certain range. For instance, when controlling an actuator, the amount of torque that it can deliver is bounded. In Fig. 3 this bounding is represented by a limiter which converts $x(t)$ values to $\tilde{x}(t)$. These values, delayed with δ time steps, are used as desired output of network A. With each iteration, the resulting output weights are used for network B. Finding a δ which corresponds to the plant dynamics, is essential in our control framework and is its main difficulty.

By applying this topology, network A is learning the controller solely on the seen plant-input and output during actual control. Network B on the contrary, uses the trained parameters to improve the control of the plant based on both the desired and actual plant response.

As mentioned before, any dynamical system with a high dimensional state representation can be used in our discussed control framework. However, to validate this framework on several tasks, we will use a Reservoir Computing network. In general, such a network can at least be applied to all tasks that can be represented by a Volterra series if the pool of network states is rich enough [24], [25] (i.e. the network is large enough). Next, we explain Reservoir Computing and the used Recursive Least Square algorithm. In Section V we evaluate our control framework on plants with different dynamical properties. These dynamics can range from linear to non-linear and from slow to fast.

III. Reservoir Computing

The RC-network model used in this paper follows the Echo State Network (ESN) approach [26]. An ESN is composed of a discrete-time recurrent neural network (i.e., the reservoir) and a linear readout layer which maps the reservoir states to the desired output. A schematic overview of this is given in Fig. 4. For many applications, the dynamics of the reservoir need to be slowed down to match the intrinsic time scale of the input data. The system's dynamics can effectively be tuned by using leaky integrator neurons [26]. Their states and the readout output are updated as follows:

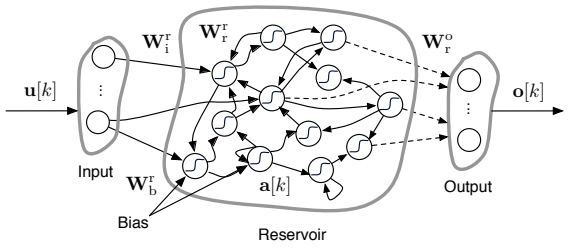

Fig. 4. Description of a Reservoir Computing network. Dashed arrows are the connections which can be trained. Solid arrows are fixed. $\mathbf{W}^{\Delta}_{*}$ is a matrix representing the connections from $*$ to Δ, where r,i,o,b denote *reservoir*, *input*, *output*, and *bias*, respectively. $\mathbf{u}[k]$, $\mathbf{o}[k]$ and $\mathbf{a}[k]$ represent the input, output and reservoir states, respectively.

$$\mathbf{a}[k+1] = (1-\gamma)\mathbf{a}[k] + \gamma \tanh\left(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{a}[k] + \mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}[k] + \mathbf{W}^{\mathrm{r}}_{\mathrm{b}}\right) \quad (1)$$

$$\mathbf{o}[k+1] = \mathbf{W}^{\mathrm{o}}_{\mathrm{r}}\mathbf{a}[k+1], \quad (2)$$

where $\mathbf{u}[k]$ denotes the input at time k, $\mathbf{a}[k]$ represents the reservoir state and $\mathbf{o}[k]$ is the output. The weight matrices $\mathbf{W}^{\Delta}_{*}$ represent the connections from $*$ to Δ between the nodes of the network (where r,i,o,b denote *reservoir*, *input*, *output*, and *bias*, respectively). All weight matrices to the reservoir (denoted as $\mathbf{W}^{\mathrm{r}}_{*}$) are initialized randomly, while all connections to the output (denoted as $\mathbf{W}^{\mathrm{o}}_{*}$) are trained using standard linear regression techniques. As nonlinearity a hyperbolic tangent function is used. After initialization, the matrix $\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}$ is normalized by dividing it with its largest absolute eigenvalue ρ, called the spectral radius. For linear neurons (no tanh-function) the spectral radius should be close, but smaller than one. Thus, operating at the edge of stability [26]. For nonlinear neurons a spectral radius around 1 can be used as an heuristic. In [27], a more analytical explanation can be found. In Equation (1) a fraction of the the previous state $\mathbf{a}[k]$ is taken into account. This operation is equivalent to a first order low-pass filter where the term γ is called the leak rate. Further investigation about time scales in reservoirs and leaky integrator neurons can be found in [28], [29]. Due to the used network model (Equation (1)) the current network output depends on a finite time window of past network inputs. Such fading memory can be related to the unique steady-state property for dynamical systems [30]. As we are using such a network to model a dynamical system, we can argue that only a dynamical system with a unique equilibrium can be controlled.

In this work all the weight matrices $\mathbf{W}^{\Delta}_{*}$ are randomly initialized according to a standard normal distribution $\mathcal{N}(0,1)$. However, $\mathbf{W}^{r}_{i}$ and $\mathbf{W}^{r}_{b}$ are scaled with the factors f^{r}_{i} and f^{r}_{b}, respectively. The number of neurons or reservoir size (denoted by N_{res}) determines the size of the connection matrix $\mathbf{W}^{r}_{r}$. The connection fraction or sparseness of this connection

matrix is described by the parameter Ψ. For choosing the number of neurons, a trade-off between execution speed and performance has to be made.

Training is performed by linear regression using the reservoir states as variables. For this, the reservoir is driven by an input sequence (the gathered plant feedback) which yields a sequence of neuron states using Equation (1) and a sequence of outputs (the input signals for the plant) using equation (2). Next, the output connections $\mathbf{W}_r^o$ are trained such that the generated output signals correspond to the desired output signals. The training can be performed offline on a fixed dataset, where desired input-output pairs are known, or online, training as more data is provided.

For online learning we use Recursive Least Squares (RLS). With each iteration the output weights are adjusted so that the network converges to the desired output:

$$\mathbf{W}_\mathbf{r}^\mathbf{o}[k] = \mathbf{W}_\mathbf{r}^\mathbf{o}[k-1] - e[k]\mathbf{P}[k]\mathbf{a}[k] \tag{3}$$

$$\mathbf{P}[0] = \frac{\mathbf{I}}{\alpha} \tag{4}$$

$$\mathbf{P}[k] = \frac{\mathbf{P}[k-1]}{\lambda} - \frac{\mathbf{P}[k-1]\mathbf{a}[k]\mathbf{a}^T[k]\mathbf{P}[k-1]}{\lambda(\lambda + \mathbf{a}^T[k]\mathbf{P}[k-1]\mathbf{a}[k])}, \tag{5}$$

with $\mathbf{a}[k]$ the current states, λ the forgetting factor and α an initially chosen value. $\mathbf{P}[k]$ is a running estimate of the Moore-Penroose pseudo inverse $(\mathbf{a}^T\mathbf{a} + \eta\mathbf{I})^{-1}$, with η a regularization parameter [31]. $\mathbf{P}[0]$ denotes the initial value of $\mathbf{P}$. The used error during training is defined as the difference between the generated output and the desired output $d[k]$:

$$e[k] = \mathbf{W}_\mathbf{r}^\mathbf{o}[k-1]\mathbf{a}[k] - d[k]. \tag{6}$$

When using RLS these output weights are rapidly and effectively modified. For this reason, RLS is also used in the FORCE approach of Sussillo and Abbott [32]. As stated before, if we want to use this learning algorithm we need to have a desired output $d[k]$, which is unknown at first. However, as described in the previous section, by initially using random values for $d[k]$ and the corresponding plant response, an inverse plant model can be trained. Each iteration the model improves resulting in a more accurate prediction of the control output.

IV. Convergence and Stability Analysis

The convergence of the error $e[k]$ in Equation (6) can be analyzed by rewriting this Equation as:

$$\mathbf{W}_\mathbf{r}^\mathbf{o}[k-1] = \frac{e[k] + d[k]}{\mathbf{a}[k]}. \tag{7}$$

As in [32], we substitute this into Equation (3) to achieve a formulation of the error after the weight update:

$$\mathbf{W}_\mathbf{r}^\mathbf{o}[k] = \frac{e[k] + d[k]}{\mathbf{a}[k]} - e[k]\mathbf{P}[k]\mathbf{a}[k] \tag{8}$$

$$\mathbf{W}_\mathbf{r}^\mathbf{o}[k]\mathbf{a}[k] = e[k] + d[k] - e[k]\mathbf{a}^T[k]\mathbf{P}[k]\mathbf{a}[k] \tag{9}$$

$$e[k] = \frac{\mathbf{W}_\mathbf{r}^\mathbf{o}[k]\mathbf{a}[k] - d[k]}{1 - \mathbf{a}^T[k]\mathbf{P}[k]\mathbf{a}[k]} \tag{10}$$

$$e_+[k] = e[k](1 - \mathbf{a}^T[k]\mathbf{P}[k]\mathbf{a}[k]). \tag{11}$$

As mentioned before $\mathbf{P}$ can be written as:

$$\mathbf{P} = (\sum_k \mathbf{a}[k]\mathbf{a}^T[k] + \alpha\mathbf{I})^{-1}. \tag{12}$$

Due to the used tanh-nonlinearity in Equation (1) and the initialization of $\mathbf{a}[k]$ we know that $|\mathbf{a}[k]| \leq \mathbf{1}$. As a results, $\mathbf{a}^T[k]\mathbf{P}[k]\mathbf{a}[k]$ in Equation (11) will change from a value close to 1 to a value that asymptotically converges to 0. Consequently, $e_+[k]$ will become small and will eventually converge to $e[k]$. At this point $\mathbf{w}[k] - \mathbf{w}[k-1]$ becomes 0.

As mentioned before, a RNN is a dynamical system which is infamously difficult to analyze. Despite the efforts made in [33] and [34], no further progress has been achieved in the quest for rigorous performance and stability guarantees. In this work however we can do the following observations concerning stability:

*1) **BIBO-stability**:* Bounded-input-bounded-output stability is guaranteed. The non-linearity in Equation (1) (e.g. $\tanh(\cdot)$) and the introduced limiter depicted in Fig. 3 ensures that the network output is bounded for all inputs.

*2) **Local stability**:* Under certain conditions, local stability at the origin can be guaranteed. We use the NLq-framework presented in [35] to derive conditions for local stability of the control system. Before we can apply this framework, we assume that $\gamma = 1$ in Equation (1). We also assume that learning has converged because a constant change in output weights would make it hard to analyze stability. Under these assumptions we only need to take Network B into account. Furthermore, we need to make sure that the applied non-linearity $y = f(x)$ fulfills the condition that for each x there exists an $h \in [0,1]$ such that $f(x) = hx$. The applied $\tanh(\cdot)$ satisfies this condition.

In this work we use the NLq framework with $q = 2$ layers where the plant is represented by a neural network interacting in a closed loop with the controller (also a neural network). By preserving the notation used in [35] we define the plant and control network as $\mathcal{M}_1$ and $\mathcal{C}_2$, respectively. According to a discrete version of the notation used in Fig. 1 and Fig. 4 both networks can be described by their neural state space models:

$$\mathcal{M}_1 : \begin{cases} \mathbf{b}[k+1] = \tanh(\hat{\mathbf{W}}_\mathbf{r}^\mathbf{r}\mathbf{b}[k] + \hat{\mathbf{W}}_\mathbf{i}^\mathbf{r}x[k] + \hat{\mathbf{W}}_\mathbf{b}^\mathbf{r}) \\ y[k] = \hat{\mathbf{W}}_\mathbf{r}^\mathbf{o}\mathbf{b}[k] + \hat{\mathbf{W}}_\mathbf{b}^\mathbf{o} \end{cases}$$

$$\mathcal{C}_2 : \begin{cases} \mathbf{a}[k+1] = \tanh(\mathbf{W}_\mathbf{i}^\mathbf{r}\mathbf{a}[k] + \mathbf{W}_\mathbf{i}^\mathbf{r}y[k] + \mathbf{W}_\mathbf{i2}^\mathbf{r}\hat{y}[k] + \mathbf{W}_\mathbf{b}^\mathbf{r}) \\ x[k] = \tanh(\mathbf{W}_\mathbf{r}^\mathbf{o}\mathbf{a}[k] + \mathbf{W}_\mathbf{b}^\mathbf{o}) \end{cases}$$

where the network weights and states of $\mathcal{M}_1$ are represented by $\hat{\mathbf{W}}_*^\Delta$ and $\mathbf{b}[k]$, respectively. The output weights $\hat{\mathbf{W}}_r^o$ are trained with RLS, the other weights are randomly initialized. $\hat{y}[k]$ denotes the desired plant output. Notice that for $x[k]$ in $\mathcal{C}_2$ the limiter in Fig. 3 is represented by the $\tanh(\cdot)$-function. After augmenting the states with $\xi[k+1] = x[k+1]$ and substituting the output/input of $\mathcal{M}_1$ with the input/output of $\mathcal{C}_2$, we can write the state space model of the entire control loop as:

$$p_{k+1} = \Gamma_1(V_1\Gamma_2(V_2p_k + B_2w_k) + B_1w_k), \tag{13}$$

with $p_{k+1} = [\mathbf{b}[k+1], \mathbf{a}[k+1], \xi[k+1]]^T$, $w_k = [\hat{y}[k], 0, 1]^T$ and both Γ_1 and Γ_2 are matrix representations of the $\tanh(\cdot)$-functions. Here we applied the same notations as in [35]. We investigate the autonomous case where no external input ($w_k = 0$) to the control loop is considered. Due to the used state space model representation, local stability at the origin is guaranteed if:

$$\rho(V_1 V_2) < 1, \tag{14}$$

with $\rho(\cdot)$ the spectral radius and $V_1 V_2$ given by:

$$V_1 V_2 = \begin{pmatrix} \hat{\mathbf{W}}_{\mathbf{r}}^{\mathbf{r}} & 0 & \hat{\mathbf{W}}_{\mathbf{i}}^{\mathbf{r}} \\ \mathbf{W}_{\mathbf{i}}^{\mathbf{r}} \hat{\mathbf{W}}_{\mathbf{r}}^{\mathbf{o}} & \mathbf{W}_{\mathbf{r}}^{\mathbf{r}} & 0 \\ \mathbf{W}_{\mathbf{r}}^{\mathbf{o}} \mathbf{W}_{\mathbf{i}}^{\mathbf{r}} \hat{\mathbf{W}}_{\mathbf{r}}^{\mathbf{o}} & \mathbf{W}_{\mathbf{r}}^{\mathbf{o}} \mathbf{W}_{\mathbf{r}}^{\mathbf{r}} & 0 \end{pmatrix}. \tag{15}$$

For instance, numerical evaluation of Equation (14) on the pitch control task in Section V gives $\rho(V_1 V_2) = 0.8925$, which implies that local stability at the origin of Equation (13) is guaranteed in this case. The size of the basin of attraction in which the controller is locally stable can be large. However as described in [36], the basin size in which local stability is proven, is equal or smaller. This basin size can be calculated by maximizing the volume of ellipsoids defined by a quadratic Lyapunov function with respect to p_k. The corresponding matrix inequalities which constrains this sequential quadratic programming problem can be found in [36]. Due to the plant dependence, the attraction basin in which local asymptotic stability is proven, needs to be calculated for each control task at hand, which is beyond the scope of this work. We refer to [36], for a more extensive description on how this can be calculated.

V. Computer Simulations and Experimental Results

To validate the designed controller we will address three tasks, each with different interesting properties. Applying a model based controller to all these tasks with different dynamical properties is impossible. In the following experiments we demonstrate that our control strategy can control different plants with different dynamics even without a predefined internal model. When a control value is used as plant-input this value is constant during the integration time. When the integration time expires, the simulation is frozen until, based on the previous plant response, a new control value is calculated. However, the calculation time needed to produce such a control value can be reduced such that it becomes smaller than the integration time. This becomes important when handling real-life and real-time applications. Though, in this work only simulations are considered.

A. Heating tank

The first control task is a process with a variable dead-time and has slow nonlinear dynamics. These control problems appear in industrial processes where measurement sensors that are used for feedback, are not integrated in the process itself (e.g. solar collector field [37]). In this experiment, as shown in Fig. 5, the system consists of a filled and constantly heated water tank with attached pipe of length L. If the output temperature of the pipe is controlled by the throughput of water that feeds the tank, this temperature depends not only on the pipe length (L) but also on the throughput itself, which is a control parameter. The fact that this parameter constantly changes (having a variable death time) has a significant impact on the performance of the control loop. Controlling such a process is a challenging task, especially without an instantaneous measurement of the process variables or with control by a delayed pump (controlling the throughput) [38], where the response on the feedback is delayed.

TABLE I
Simulation parameters for the heating tank model

Parameter	Value	Parameter	Value
c_p	4186 J/kgK	K_{tube}	0.99
$V_{tank} = LS$	1.13 l	T_s	4 s
Q	1100 J	$q(0)$	0.0167 l/s
ρ	1 kg/l	$q(t)$	$\in [0.005, 0.03]$ l/s
T_{in}	15 °C		

*1) **Model**:* The dynamics of the plant model illustrated in Fig. 5 are described by the following nonlinear differential equation:

$$\rho c_p V_{tank} \frac{dT_{tank}(t)}{dt} = Q + \rho c_p q(t)(T_{in} - T_{tank}(t)), \tag{16}$$

where c_p denotes the specific heating capacity of water, V_{tank} the volume of the tank, $T_{tank}(t)$ the water temperature in the tank, T_{in} the temperature of the added water, Q the added heat, ρ the density of water and $q(t)$ the throughput of the added water. The dynamics of the outlet pipe with length L and area S are modeled by the following low-pass filter:

$$\frac{T_{tube}(s)}{T_{tank}(s)} = \frac{K_{tube}}{T_{tube} s + 1}, \tag{17}$$

where K_{tube} is the fraction of temperature change from tank to tube and T_{tube} an unmeasurable temperature with temperature T_{out} that follows the equation:

$$T_{out}(t) = T_{tube}(t - d(t)). \tag{18}$$

In the previous equation $d(t)$ describes the variable dead-time which equals:

$$d(t) = T_s N_d = \frac{LS}{q(t)}, \tag{19}$$

where T_s represents the sampling period. N_d describes the unknown dead-time. It is clear that by knowing L, S and $q(t)$ the variable dead-time can be calculated. In our simulation we used the parameters given in Table I. For simulation we use the Dormand-Prince method [39], also known as Runga-Kutta (4,5), with an integration time step of 4 s.

*2) **Controller**:* To control this plant we use a Reservoir Computing network in the proposed control framework described in Section II-A. The limiter bounds the throughput $x(t) = q(t)$ to the allowed values for $q(t)$ shown in Table I. Next, we train the output connections of network A with the values $\tilde{x}(t - \delta)$. The feedback values $y(t) = T_{out}(t)$

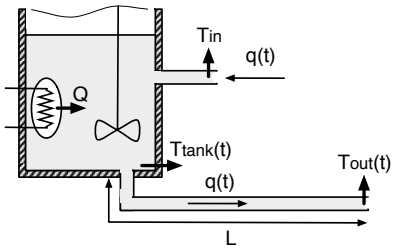

Fig. 5. Illustration of the heating tank process. A tank filled with water is constantly heated with heat flow Q. Water with temperature T_{in} is pumped in and exits the tank with a throughput $q(t)$. The water that is leaking from the tank has a certain amount of time to cool down in a pipe with length L before measurement of temperature T_{out}. To insure homogeneous heating, the water in the tank is stirred. The control task: How to change the throughput $q(t)$ to get a desired temperature T_{out}?

TABLE II
NETWORK PARAMETERS OF HEATING TANK TASK

Parameter	Value	Parameter	Value
N_{res}	500 neurons	f_i^r	0.1
ρ	1	f_b^r	0.5
Ψ	50 %	γ	0.5
δ	30		

from the plant are given to the networks in a normalized form (subtracted with the mean an divided by its standard deviation). The used parameters of both networks, shown in Table II, were optimized by performing grid search on a validation set (target temperatures forming a staircase signal).

The introduced RLS-parameters defined in Section III are set to $\lambda = 1 - 10^{-6}$ and $\alpha = 10$. The initial output weights $\mathbf{w}(0)$ are normalized random values ($\mathcal{N}(0,1)$).

*3) **Results**:* For our simulation we have applied the controller to the described simulation model for 12000 time steps or 13.33 hours real time. The desired response of the plant consists of different phases where we try, in the first phase, to control the plant to have a $y(t)$ that changes relatively quickly. In this phase, we use red noise by feeding white noise through a low-pass filter. Afterwards, this noise is scaled to represent realistic temperature values. The second phase consists of a staircase signal. Both phases are randomly generated for each experiment. The first 6000 time steps of the experiment are shown in Fig. 6. One can see, by looking at the average quadratic change in output weights, that the proposed controller is learning to control the plant within the first 2000 time steps. In Fig. 7 the transition to a staircase signal is shown. Here the controller is able to adapt by changing its output weights according to the desired plant output. As shown at the bottom of both Fig. 6 and 7, the generated throughput during this staircase signal, is close to an optimal control signal. Indeed, a temperature decrease is generated by setting the throughput very high in the beginning and lowering it afterwards.

We compared the proposed controller with a model based controller, called the Nonlinear Predictive Control Strategy (NEPSAC), that out-preforms more classical approaches

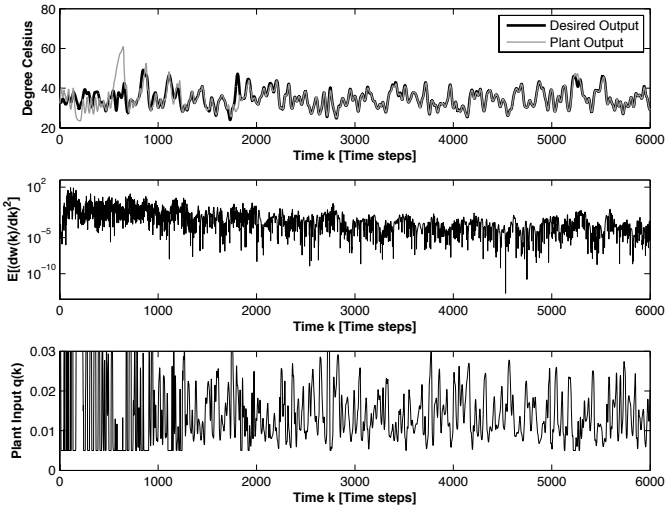

Fig. 6. Overview of the first 6000 time steps of the simulation. Here, the desired output is an always changing temperature. Above, the actual plant-output (shifted over δ) together with the desired one are shown. In the middle, the average quadratic weight adaptation is illustrated. At the bottom, the actual plant-input, which is generated by the controller is shown.

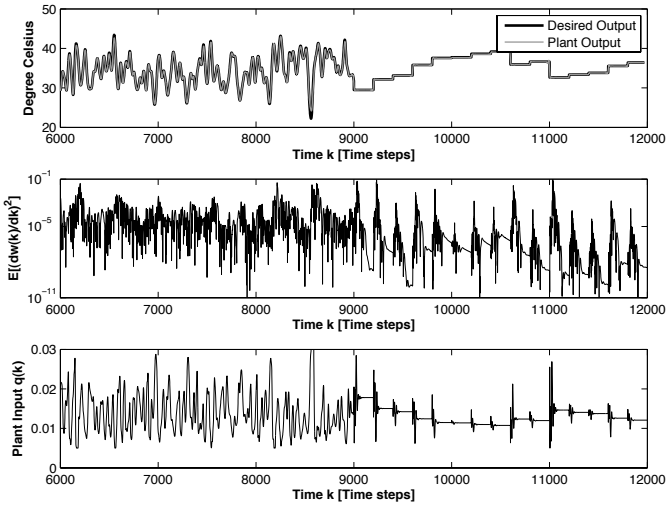

Fig. 7. Overview of the last 6000 simulation time steps. Here, the desired output is a constantly variating temperature which eventually shifts to a desired temperature profile according to a staircase. The middle plot illustrates the average quadratic weight adaptation of the output weights. The bottom plot shows the generated control signal.

(such as PID) on this task [40], [41]. For comparison, a staircase signal is used as desired plant output which, after 2000 time steps of initialization, is shown in Fig. 10.

We notice that both our implementation of the NEPSAC and the proposed controller have some trouble in the beginning due to the transition between the faster variation in output temperature and the staircase signal. Afterwards, both are able to follow the desired temperature. Taking a closer look at the staircase in Fig. 11 between time steps 5000 and 5800 reveals that, after a temperature change, the proposed controller is able to reach the desired temperature faster than NEPSAC. The time to reach the desired output temperature

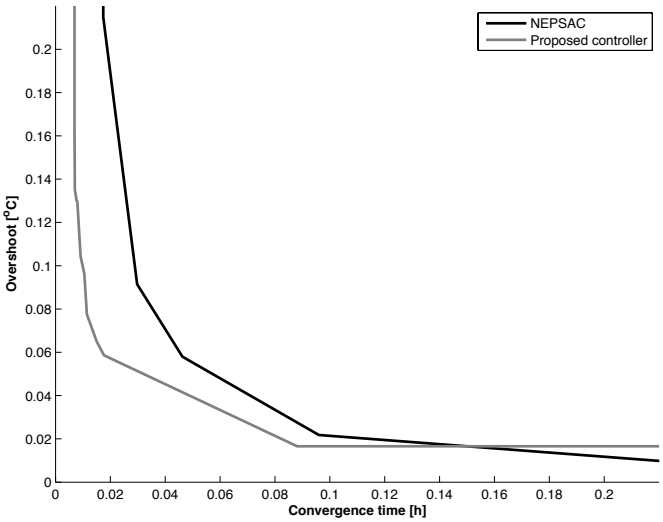

Fig. 8. This plot shows the Pareto front for both the proposed and the NEPSAC controller. When the overshoot is allowed to be equal or larger than $0.016°C$, the proposed controller is preferred. For a smaller overshoot but a larger convergence time NEPSAC is better in the comparison.

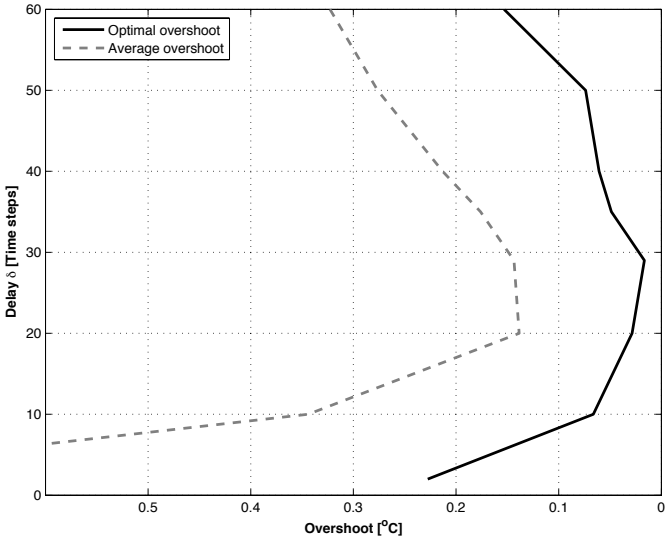

Fig. 9. Illustration of the effect of the delay δ on the optimal and average overshoot of the proposed controller. The average overshoot is calculated over all experiments with the same delay δ but with different leak rates and input scaling. Increasing δ will improve the convergence time until the delay becomes larger than the memory capacity of the RC-network. The memory capacity of an RC-network with 500 neurons starts to decrease around 30 time steps after which it will decrease dramatically.

is called converging time.

We evaluate the convergence time as the time needed to approach the set point after which it stays within a predefined margin around this set point. In the following experiments for this control task we have set this margin to $0.01\,°\mathrm{C}$. The overshoot is measured as being the largest difference between the desired set point and the produced plant output after the set point has changed.

For NEPSAC the balance between overshoot and convergence time is regulated by its prediction horizon $\in [5,\ldots,50]$ which is illustrated by its Pareto front [42] in Fig. 8. The larger its prediction horizon the smaller the overshoot but with the disadvantage that the convergence time increases.

As mentioned before, δ defines a time window with which the dynamics of the plant are observed. If δ is small, the learned model is more sensitive to fast dynamical changes, and vice versa. The used leak rate on the other hand, basically implements a low-pass filter on the state changes. As a results, the overshoot/convergence-time balance for the proposed controller is depending on both the value of the delay δ and the leak rate γ. In Fig. 8 the Pareto front for the optimal $\delta = 29$ and averaged over 5 reservoirs is given. This means that the defined balance in this front is controlled by its leak rate and input scaling. For a defined input scaling, increasing the leak rate will lower the overshoot and increases the convergence time. However, experiments show that the choice of the delay δ influences the resulting Pareto front as well. Not only is δ depending on the rate at which relevant samples are presented to the network but also on the memory capacity of the network itself. This is shown in Fig. 9. For different delays the optimal/lowest and average[1] overshoot is shown. One can notice the improvement in overshoot by increasing the delay δ until the delay becomes larger than the memory capacity[2] of the network (around 30 time steps for a reservoir with 500 neurons). Increasing the delay δ further will lead to a larger overshoot.

Now, if we compare both Pareto fronts we can conclude that the proposed controller is more suitable for tasks where fast convergence is needed and the overshoot is allowed to be larger than $0.016\,°\mathrm{C}$. For slower control where the desired overshoot is lower and a convergence time of 0.15 hours or larger is allowed, NEPSAC is the optimal choice.

B. Pitch Control

The second task we consider is taken from a set of control examples [43]. The purpose of this task is to control the pitch of a simplified aircraft model by changing the elevator deflection angle. However, changing this deflection angle causes the pitch angle to move slowly. The time needed to reach the desired angle depends on the distance between the previous and current pitch angle which makes this task nontrivial.

*1) **Model**:* The pitch control problem is simplified by assuming a steady cruise of the aircraft at constant velocity V and altitude. Under these conditions the control problem can be formulated as:

$$\frac{d\alpha}{dt} = -0.313\alpha + 56.7q + 0.232\eta$$

$$\frac{dq}{dt} = -0.0139\alpha - 0.426q + 0.0203\eta$$

[1]Average over all experiments with the same δ but with different leak rates and input scaling.

[2]The number of time steps, traces of past inputs are reflected in the current states (length of short-term memory).

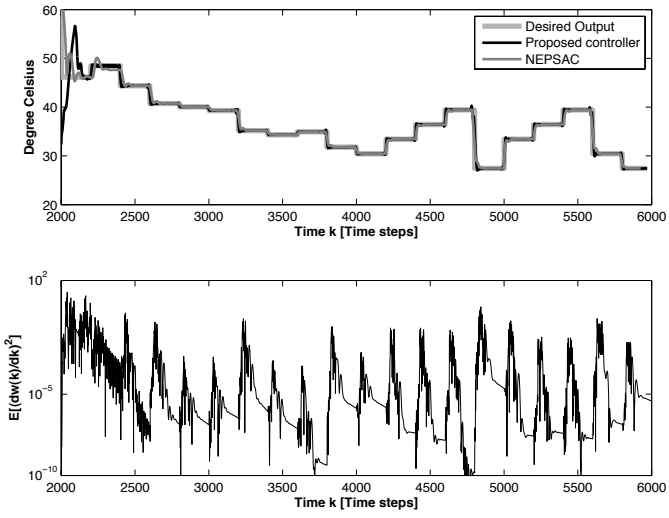

Fig. 10. The actual plant output is shown for the proposed controller (shifted over δ) and for the NEPSAC-controller. This gives, together with the desired plant output, a good representation of the control performance.

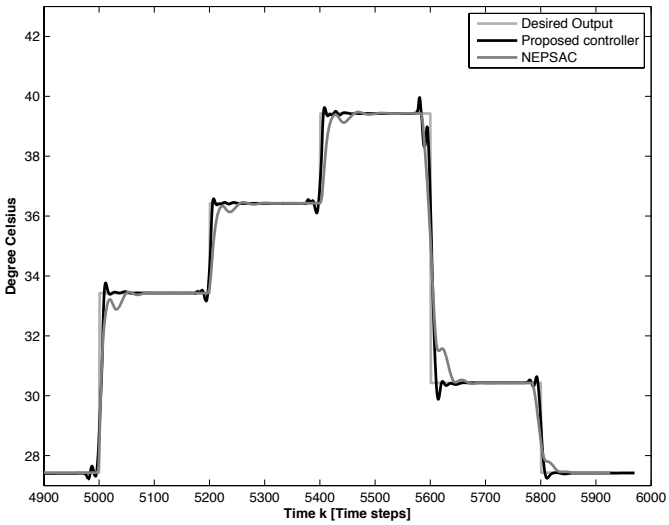

Fig. 11. This plots shows a more detailed view of the overshoot and convergence time of both controllers.

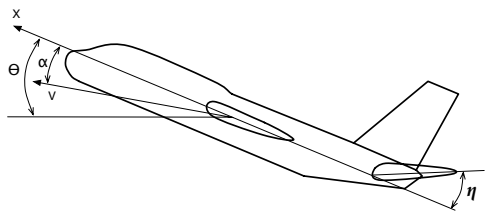

Fig. 12. The representation of the aircraft which is used in simulation. V represents the velocity vector and X denotes the center axis of the plain. Θ, α and η on the other hand represent the pitch, attack and elevator deflection angle, respectively. The control task: How to control η to achieve a desired pitch angle Θ?

$$\frac{d\Theta}{dt} = 56.7q.$$

As shown in Fig. 12, α is describing the angle of attack, q the pitch rate, η the elevator deflection angle and Θ the pitch angle. For simulation we again use the Runga-Kutta (4,5) method with an integration time step of 50 ms.

TABLE III
NETWORK PARAMETERS OF PITCH CONTROL TASK

Parameter	Value	Parameter	Value
N_{res}	500 neurons	f_i^r	0.2
ρ	0.7	f_b^r	1
Ψ	100 %	γ	1
δ	10		

*2) **Controller**:* The dynamics of the simplified control task are linear. However, they present some interesting challenges for the proposed learning algorithm. Changing the elevator angle η causes the angle of attack α to change slowly until it settles. When η is changed again before α has settled, the current angle of attack is partially depending on its previous angle. As mentioned before the pitch of the aircraft is controlled by changing the elevator deflection angle. Therefore, the proposed controller, shown in Fig. 3, has an input and output which is defined as $y = \Theta$ and $x = \eta$, respectively. In Table III the network parameters for both network *A* and B are shown. All these parameters were determined by performing a grid search on a validation set.

*3) **Results**:* For the evaluation of the controller we conducted experiments where a desired pitch angle is set. After keeping the target pitch angle constant for 300 time steps, the target pitch angle is changed to another value. These values are randomly chosen according to a standard normal distribution: $\widehat{y}(t) \in \mathcal{N}(0, 0.35)$ rad. Each experiment takes 10000 time steps. To evaluate the control performance we compare it with a controlled method called Linear Quadratic Regulator (LQR) [1], [44]. This method allows us to find (tuning a weighting factor p) an optimal control matrix $\mathbf{K}$ that results in a appropriate state-feedback controller $\eta = -\mathbf{K}\chi$ (χ represents the controller state $[\alpha, q, \Theta]^T$). In [43] and LQR controller design is presented which we will use and which results in a gain vector $K = [-0.6435, 169.6950, 7.0711]$ with a weighting factor $p = 50$.

As shown in Fig. 13, the proposed controller's performance improves as the experiment progresses. The learned controller is changing the deflector angle $x = \eta$ fast after a set point adjustment. Afterwards, as the pitch angle $y = \Theta$ converges, the generated output converges to 0 rad. In Fig. 14 a more detailed section of such an experiment is shown ($\delta = 3$ and $\gamma = 0.95$). Here, the difference between both controllers and the desired plant output is clearly visible. The learned controller causes the pitch angle to change rather fast before approaching the desired set point. After almost no overshoot, the resulting pitch angle will converge. This small overshoot is clearly less than the overshoot of the LQR approach. However, as in most control tasks, a trade-off between overshoot and convergence time has to be made. To insure a good comparison of both metrics the design requirements used to design the LQR-controller should be the same as the ones used for the proposed controller. However,

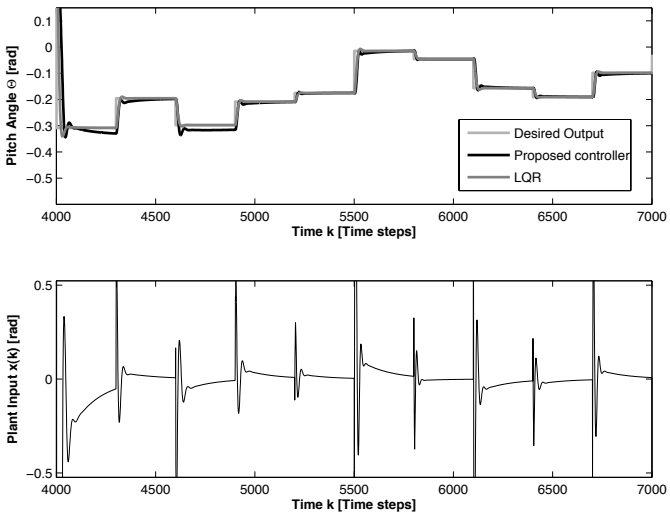

Fig. 13. The top plot shows a part of a pitch control experiment where the desired pitch angle is compared with the ones acquired by using the proposed controller and the Linear Quadratic Regulator approach. It demonstrates an improvement in the convergence of the proposed controller as the experiments progresses. The bottom plot shows the corresponding plant input (elevator angle η) produced by the proposed controller.

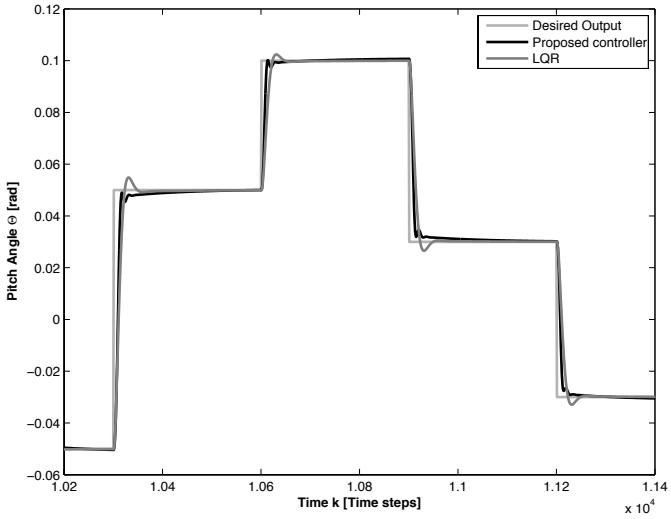

Fig. 14. This plot gives a detailed view of the proposed controller's performance at the end of an experiment. Furthermore, the plant-output under influence of the LQR approach is shown which illustrates the difference in convergence time and overshoot between both approaches.

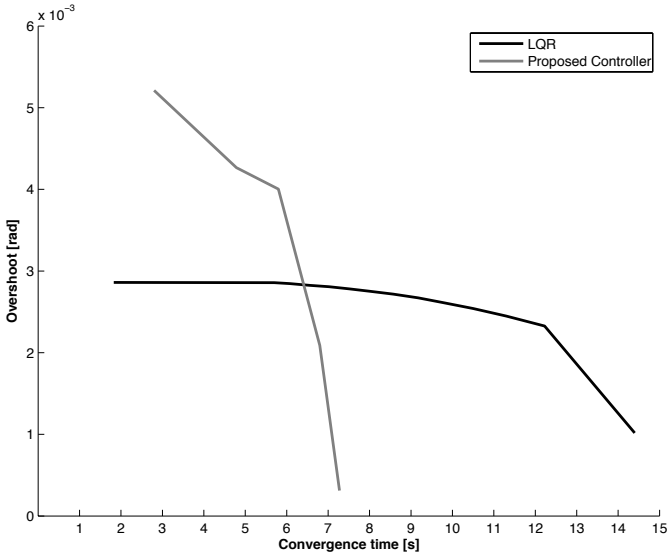

Fig. 15. This plot shows the Pareto front of both the proposed controller and the LQR-approach, given the predefined margin conditions (error $<$ 0.0005 rad). It illustrates that the LQR-controller performs better as long as a convergence time of less than 6.5 s is required. As soon as a the pitch angle is allowed to converge slower, the proposed controller is recommendable. Furthermore, most of the results of the LQR approach converge fast. Only when the weighting factor p becomes large, the convergence time increases drastically.

by creating a Pareto front of both approaches we can evaluate and compare the control performance more thoroughly.

As in the previous task we calculate the overshoot but use a margin of 0.0005 rad to determine the convergence time.

The overshoot and convergence time of the proposed controller was calculated for different parameters values of $\gamma \in [0.6, \ldots, 1]$ and $\delta \in [2, \ldots, 12]$. A large γ results in a smaller overshoot than a smaller γ. Consequently, the convergence time will be larger with a large γ than with a small γ. Similarly as for the proposed controller, we calculated both the overshoot and convergence time of the LQR approach for multiple weighting factors $p \in [3, \ldots, 150]$. These experiments result in the Pareto front shown in Fig. 15.

The Pareto front illustrates that the proposed controller is performing worse than LQR when a small convergence time is needed (< 6.5 s). However, when a larger convergence time is allowed the resulting overshoot of our controller is much smaller (Fig. 14). Therefore, under these conditions, our controller is more appropriate for use than the LQR approach. As the LQR approach is fully deterministic the results of the conducted parameter sweep (changing weighting factor p) are all located near the Pareto front. With LQR a small overshoot and a lower convergence time is possible for smaller weighting factors ($p < 100$). The Pareto front of the proposed controller is calculated by averaging results over 6 RC-networks.

C. Double inverted pendulum

The balancing task of a double inverted pendulum is a well known task in control theory and presents some interesting control challenges. Here, 2 rods connected with a joint need to be balanced in a upright position by only controlling the angle of one of the rods. In a small region around this desired position the dynamics are approximately linear. However, outside this region the dynamics of the pendulum are strongly nonlinear. In this work we only consider the pendulum stabilization and not the swing-up.

*1) **Model**:* The double inverted pendulum is modeled as illustrated in Fig. 16. In this model the weight of the rods is neglected and each end of the rod is modeled as a point mass. The Cartesian coordinates of these point masses are

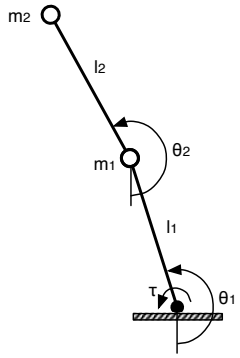

Fig. 16. The representation of the double pendulum which is used as simulation model. The control task: How to drive the torque of the motor to achieve balancing of the double inverted pendulum?

TABLE IV
SIMULATION PARAMETERS FOR THE DOUBLE PENDULUM

Parameter	Value	Parameter	Value
m_1	1 kg	θ_1	$\in [\frac{\pi}{2}, \frac{3\pi}{2}]$
m_2	1 kg	θ_2	$\in [\frac{\pi}{2}, \frac{3\pi}{2}]$
l_1	1 m	τ_{max}	50 N
l_2	1 m	Time step	50 ms

given by (x_1, y_1) for m_1 and (x_2, y_2) for m_2, with:

$$\begin{aligned} x_1 &= l_1 \sin(\theta_1) \\ y_1 &= -l_1 \cos(\theta_1) \\ x_2 &= l_1 \sin(\theta_1) + l_2 \sin(\theta_2) \\ y_2 &= -l_1 \cos(\theta_1) - l_2 \cos(\theta_2). \end{aligned} \tag{20}$$

Using these equations the potential energy V and kinetic energy T can be derived:

$$\begin{aligned} V &= m_1 g y_1 + m_2 g y_2 \\ T &= \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2, \end{aligned} \tag{21}$$

where $v_i = \frac{dx_i}{dt} + \frac{dy_i}{dt}$. To validate the model one can compute the total energy $E = V + T$, which should be a constant over time when the applied torque is zero.

Next, we use the Lagrangian transformation with $L = T - V$ and define the applied torque τ by using the Euler-Lagrange differential equation:

$$\frac{\partial L}{\partial \theta_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) = \tau_i, \tag{22}$$

with $\tau = \tau_1$ and $\tau_2 = 0$ because we only apply torque on the first rod. By writing this equation in function of θ_i and $\frac{d\theta_i}{dt}$ for $i = 1, 2$ one can solve this system with respect to $\frac{d\dot{\theta}_i}{dt}$, which leads to the equations of motion for the double pendulum (Fig. 16). Similar as in the previous tasks we use the Runga-Kutta (4,5) method with an integration time step op 50 ms.

*2) **Controller**:* This control task is different from the other tasks in the sense that the proposed controller in this task is limited in the number of examples and the amount of time given to learn. In most cases an initial control effort does not succeed in balancing which causes the rods to fall down quite easily. The amount of information that can be used for learning is therefore limited. Learning can easily lead to an unbalanceable position of the rods. As a result, balancing the pendulum means that the range of a possible solution is restricted for θ_1 and θ_2 (shown in Table IV). This region consists of both a linear and nonlinear dynamic region of the pendulum. We assume that the controller is unable to control the pendulum when the pendulum exceeds the restricted range and, when this happens the simulation is reset. Each balancing trial until a simulation reset, we call an episode. A simulation reset implies randomly initializing both rod positions within the regions for θ_1 and θ_2, holding this initial position for δ time steps and reinitializing the network states to their original values.

TABLE V
NETWORK PARAMETERS OF DOUBLE INVERTED PENDULUM TASK

Parameter	Value	Parameter	Value
N_{res}	300 neurons	f_i^r	1
ρ	1	f_b^r	0.5
Ψ	100 %	γ	1
δ	1		

For this task the proposed controller, shown in Fig. 3, has an input and output which are defined as $\mathbf{y}(t) = [\theta_1, \theta_2]$ and $x(t) = \tau$, respectively. The angle θ_2 of the second rod is scaled up with an experimentally determined value of 10 by applying grid search. A small change of θ_2 will have a larger influence on the network than a small change of θ_1. As a result the network will first prioritize the stabilization of θ_2 before stabilizing θ_1 to the desired angle. The output of network B $x(t)$ is limited by a limiter to $\tilde{x}(t)$ by insuring $|x(t)| < \tau_{max}$. The network parameters used for both networks are shown in Table V. All parameters were optimized by applying a grid search. The introduced RLS-parameters are set to $\lambda = 1 - 10^{-6}$ and $\alpha = 1$. The initial output weights $\mathbf{w}(0)$ are normalized random values ($\mathcal{N}(0,1)$).

*3) **Results**:* Each conducted experiment takes 50000 time steps or 41.7 minutes. For each episode, both rods are randomly initialized to a value in $[\pi - 0.15, \pi + 0.15]$.

Fig. 17 shows such an experiment where balancing to the desired upright position was achieved after 8 episodes. The end of an episode is indicated by a vertical dotted line. We notice an increase in episode duration due to the learning progress of the proposed controller. In the 9th and final episode balancing is achieved. However, this does not imply that the acquired controller will be able to balance the pendulum given any initial position of the rods. As the desired set point for both angles is constant, the acquired data points used for training are restricted to the amount of episodes needed to achieve stabilization. In its final episode the controller only learns how to keep the pendulum in the upright position which eventually converges to one constant action-observation pair. This pair will not further improve the internal model representation of the pendulum. The learned

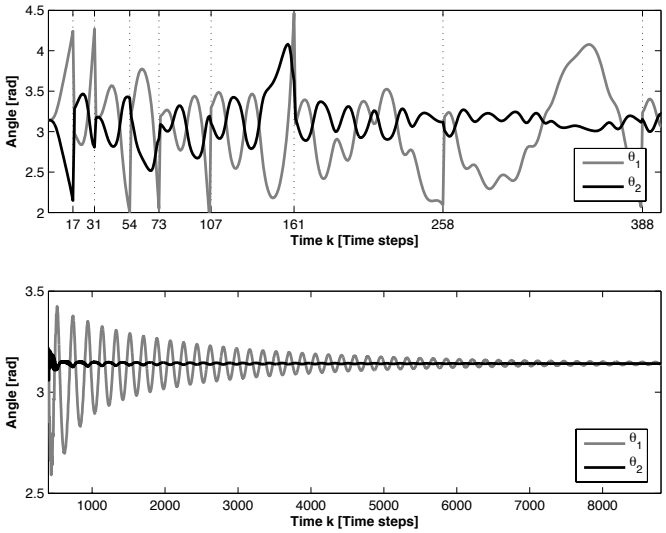

Fig. 17. Both plots illustrate the learning progress of the controller in the double inverted pendulum task. The top plot shows the first efforts of trying to balance the pendulum. Each vertical dotted line marks the beginning of a new episode at which the pendulum states are reset. In this experiment 8 episodes are necessary to successfully balance the pendulum in the 9th episode. The continuation of this last episode is shown in the bottom plot.

internal model is sufficient to balance the pendulum given the initial conditions of the episode. Comparing the acquired control with another model based controller is thus, due to the nature of the proposed controller and the task, not useful.

Although no convergence is achieved is some episodes, they can take a long time because of the good balancing efforts of the controller. As a result, some experiments finish before actual convergence to the upright position emerged. In our evaluation of the controller we assumed convergence when the errors on both angles clearly become smaller in the last episode. The number of episodes needed to achieve balancing averaged over 40 experiments is 13.75 episodes.

Furthermore, one can notice, due to the chosen input scaling, that the pendulum is indeed balanced by first prioritizing the convergence of θ_2 and afterwards θ_1.

VI. Conclusions

In this work we presented an novel feedback control framework. The core of this framework is a dynamical system, referred to as network A or B, which state representation is sufficiently rich (e.g. recurrent neural network) to hold an inverse model of the plant. The excitation of network B is used to generate a plant-input and eventually a plant-response. Afterwards, this pair is used to train the output weights of a network A. These weights in turn are used as the output weights of network B (weight sharing). Each iteration network A improves its representation of an inverse model of the dynamical system (plant). As both networks are identical (except for their input), the controlling performance of network B, which has the desired plant-output as network input, will improve. By applying this framework, accurate control on a wide variety of plants is achieved fast and online without the need for a pre-acquired plant model. Furthermore, we analyzed the convergence of the training algorithm and presented a method which allows the stability analysis under which local asymptotic stability is guaranteed.

The proposed control framework was validated on several challenging control tasks with different dynamics by using Reservoir Computing networks as learning modules: the heating tank (slow nonlinear dynamics), flight pitch control (slow linear dynamics) and the double inverted pendulum (fast linear and nonlinear dynamics). In the conducted experiments, we compared the proposed controller to other standard control techniques.

The results of the heating tank experiments show that the proposed controller is able to react relatively quickly to changes in the desired plant-output. The tracking of different kinds of output signals (red noise signal and staircase signal) was demonstrated. Although such a varying desired output improves plant exploration, a constant desired plant output can be handled as well. The performance was compared with an existing state-of-the-art model-based control method, NEPSAC. In this comparison we have shown that the proposed controller converges faster, when a moderate overshoot is allowed.The disadvantage of using NEPSAC is its slower control and, as a results, its longer convergence time. The introduced delay δ depends on the rate at which relevant samples are presented to the network. Slow dynamics need a larger δ, fast dynamics need a smaller delay. Furthermore, we have found that in this task the improvement in overshoot by increasing the introduced delay δ is also limited by the memory capacity of the used network. This observation can be argued by the fact that the modeling power of the relation between $(y(t-\delta), y(t)$ and $\tilde{x}(t-\delta))$ is lost when the RC-network is unable to "remember" it δ time steps later.

During the flight pitch control experiments the controller needs to switch between different desired pitch angles by controlling the elevator deflection. Due to the online learning nature of the controller, the acquired model representation of the plant improves as more samples are presented. As a result, the controller's performance increases as the experiment progresses. Furthermore, we compared the controller's performance with a classical LQR-controller. As for the previous task, the proposed controller is the most accurate if moderate convergence times are allowed.

The double inverted pendulum balancing task is different from the two other tasks, because the pendulum can be controlled outside of its controllable region. Due to the online learning nature of the proposed controller, this needs to be appropriately managed. By reinitializing the network states and the double pendulum, the resulting control was shown to be successful.

Currently we are applying this framework on real-life systems such as a quadruped robot. However, this is research in progress and beyond the scope of this work. Other future work could evaluate the importance of the learning module by comparing to other approaches than RC. Furthermore, an automated selection of the δ parameter is desirable.

ACKNOWLEDGMENT

This work was partially funded by a Ph.D. grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) and the FP7 funded AMARSi EU project under grant agreement FP7-248311.

REFERENCES

[1] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. Wiley-Interscience New York, 1972, vol. 188.

[2] F. Lewis, D. Vrabie, and V. Syrmos, *Optimal control*. Wiley, 2012.

[3] B. Cessac, "A view of neural networks as dynamical systems," *Arxiv preprint arXiv:0901.2203*, 2009.

[4] M. Kawato, K. Furukawa, and R. Suzuki, "A hierarchical neural-network model for control and learning of voluntary movement," *Biological Cybernetics*, vol. 57, no. 3, pp. 169–185, 1987.

[5] K. Narendra and K. Parthasarathy, "Adaptive identification and control of dynamical systems using neural networks," in *Proceedings of the 28th IEEE Conference on Decision and Control*, 1989, pp. 1737–1738.

[6] D. Nguyen and B. Widrow, "The truck backer-upper: an example of self-learning in neuralnetworks," in *International Joint Conference on Neural Networks (IJCNN)*, 1989, pp. 357–363.

[7] L. Hung and H. Chung, "Decoupled control using neural network-based sliding-mode controller for nonlinear systems," *Expert Systems with Applications*, vol. 32, no. 4, pp. 1168–1182, 2007.

[8] J. Spooner and K. Passino, "Stable adaptive control using fuzzy systems and neural networks," *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 3, pp. 339–359, 1996.

[9] S. Ge, C. Yang, and T. Lee, "Adaptive predictive control using neural network for a class of pure-feedback systems in discrete time," *IEEE Trans. Neural Netw.*, vol. 19, no. 9, pp. 1599–1614, 2008.

[10] C. Yang, S. Ge, C. Xiang, T. Chai, and T. Lee, "Output feedback NN control for two classes of discrete-time systems with unknown control directions in a unified approach," *IEEE Trans. Neural Netw.*, vol. 19, no. 11, pp. 1873–1886, 2008.

[11] A. Levin and K. Narendra, "Control of nonlinear dynamical systems using neural networks. ii. observability, identification, and control," *IEEE Trans. Neural Netw.*, vol. 7, no. 1, pp. 30–42, 1996.

[12] H. Su and T. McAvoy, "Artificial neural network for nonlinear process identification and control," in *Nonlinear process control*. Prentice-Hall, Inc., 1997, pp. 371–428.

[13] Y. Pan and J. Wang, "Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks," *IEEE Trans. Ind. Electron.*, no. 99, pp. 1–1, 2011.

[14] D. Prokhorov, "Training recurrent neurocontrollers for real-time applications," *IEEE Trans. Neural Netw.*, vol. 18, no. 4, pp. 1003–1015, 2007.

[15] C. Wang and D. Hill, "Learning from neural control," *IEEE Trans. Neural Netw.*, vol. 17, no. 1, pp. 130–146, 2006.

[16] ——, "Deterministic learning and rapid dynamical pattern recognition," *IEEE Trans. Neural Netw.*, vol. 18, no. 3, pp. 617–630, 2007.

[17] T. Chow and Y. Fang, "A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics," *IEEE Trans. Ind. Electron.*, vol. 45, no. 1, pp. 151–161, 1998.

[18] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, 1994.

[19] B. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1212–1228, 1995.

[20] J. Suykens, B. D. Moor, and J. Vandewalle, "Toward optical signal processing using photonic reservoir computing," *Optics Express*, vol. 16, no. 15, pp. 11 182–11 192, 2008.

[21] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, p. 78, 2004.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] H. Jaeger, "A method for supervised teaching of a recurrent artificial neural network," WO Patent Application WO 2002/031 764 A2, 04 18, 2002.

[24] W. Maass, "Liquid state machines: Motivation, theory, and applications," in *Computability in Context: Computation and Logic in the Real World*, B. Cooper and A. Sorbi, Eds. Imperial College Press, 2010, pp. 275–296.

[25] S. Boyd and L. Chua, "Fading memory and the problem of approximating nonlinear operators with volterra series," *IEEE Trans. Circuits Syst.*, vol. 32, no. 11, pp. 1150–1161, 1985.

[26] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks," Technical Report GMD Report 148, German National Research Center for Information Technology, Tech. Rep., 2001.

[27] B. Zhang, D. Miller, and Y. Wang, "Nonlinear system modeling with random matrices: Echo state networks revisited," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 1, pp. 175–182, 2012.

[28] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.

[29] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007*, 2007, p. 471.

[30] L. Chua and D. Green, "A qualitative analysis of the behavior of dynamic nonlinear networks: Steady-state solutions of nonautonomous networks," *IEEE Trans. Circuits Syst.*, vol. 23, no. 9, pp. 530–550, 1976.

[31] R. Penrose, "A generalized inverse for matrices," in *Mathematical proceedings of the Cambridge philosophical society*, vol. 51, no. 03. Cambridge University Press, 2008, pp. 406–413.

[32] D. Sussillo and L. Abbott, "Generating coherent patterns of activity from chaotic neural networks," *Neuron*, vol. 63, no. 4, pp. 544–557, 2009.

[33] I. Tyukin, D. Prokhorov, and V. Terekhov, "Adaptive control with nonconvex parameterization," *IEEE Trans. Autom. Control*, vol. 48, no. 4, pp. 554–567, 2003.

[34] N. Barabanov and D. Prokhorov, "A new method for stability analysis of nonlinear discrete-time systems," *IEEE Trans. Autom. Control*, vol. 48, no. 12, pp. 2250–2255, 2003.

[35] J. Suykens, J. Vandewalle, and B. De Moor, "Nlq theory: checking and imposing stability of recurrent neural networks for nonlinear modeling," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2682–2691, 1997.

[36] J. Suykens, B. De Moor, and J. Vandewalle, "Robust local stability of multilayer recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 11, no. 1, pp. 222–229, 2000.

[37] B. Torrico, L. Roca, J. Normey-Rico, J. Guzman, and L. Yebra, "Robust Nonlinear Predictive Control Applied to a Solar Collector Field in a Solar Desalination Plant," *IEEE Trans. Control Syst. Technol.*, no. 99, pp. 1–10, 2010.

[38] J. Richard, "Time-delay systems: an overview of some recent advances and open problems," *Automatica*, vol. 39, no. 10, pp. 1667–1694, 2003.

[39] J. Dormand and P. Prince, "A family of embedded runge-kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

[40] M. Gálvez-Carrillo, R. De Keyser, and C. Ionescu, "Nonlinear predictive control with dead-time compensator: Application to a solar power plant," *Solar Energy*, vol. 83, no. 5, pp. 743–752, 2009.

[41] R. De Keyser, "Model Based Predictive Control, Invited Chapter in UNESCO Encyclopaedia of Life Support Systems (EoLSS)," 2003.

[42] J. Horn, N. Nafpliotis, and D. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Proc. of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence*. IEEE, 1994, pp. 82–87.

[43] B. Messner and D. Tilbury, "Digital Control Tutorial by University of Michigan and Carnegie Mellon," 1996.

[44] E. Sontag, *Mathematical control theory: deterministic finite dimensional systems*. Springer-Verlag, 1998, vol. 6.